LEARNING WHERE TO STORE: QUERY-AWARE IVF INDICES

Anonymous authors
Paper under double-blind review

ABSTRACT

Approximate nearest neighbor (ANN) vector search is an essential primitive in modern AI and ML applications, e.g., to support retrieval augmented generation. For very large datasets, one of the most popular ANN methods is the inverted-file (IVF) index, which performs locality-friendly contiguous scans, can run on GPUs and on disk, and can serve as the first filtering step in ANN search pipelines. IVF clusters vectors by k-means and stores each vector in the list of its nearest centroid. Achieving high recall typically requires scanning more clusters; an alternative is to place copies of a vector in several lists. We present *query-aware IVF*, which learns where to store copies from a sample of the query distribution, formulating the allocation as a maximum coverage problem that we solve greedily. On the 24 datasets of the Vector Index Benchmark for Embeddings (VIBE), at recall@100 = 0.9, our query-aware index is faster than the IVF baselines on all 24 datasets: a median 1.63× faster than the best query-agnostic replication and up to 21.9× faster, and a median 1.79× faster than vanilla IVF on the 21 datasets where vanilla IVF reaches that recall, at a median 2.58× the memory of vanilla IVF.

1 INTRODUCTION

Vector search has become a basic primitive of the artificial intelligence stack, where documents, images, code, and user interactions are embedded into vector spaces, such that semantic similarities and dissimilarities are captured geometrically. The underlying algorithmic task (enabling retrieval-augmented generation (Gao et al., 2023), semantic search, recommendation, entity resolution (Zeakis et al., 2023)) is the nearest neighbor search problem: given a dataset of vectors, build an index so that, on new query vectors, the index can quickly find its nearest neighbors in the dataset. Among the many production algorithms for (approximate) nearest neighbor (Douze et al., 2026; Wang et al., 2021; Upreti et al., 2025), the inverted-file (IVF) index (Sivic & Zisserman, 2003; Jégou et al., 2011) is a particularly simple and appealing one: IVF partitions the dataset into lists using k-means, and answers a query by scanning the few lists whose centroids are closest to it. Scans are contiguous and enable streaming reads rather than pointer chasing; the same index can be served from RAM, from disk (Chen et al., 2021) or from GPUs (Johnson et al., 2021); updating the dataset is simply an addition or deletion; and the index readily supports "filtered" search (Simhadri et al., 2024). IVF is generally preferable to other methods (e.g., HNSW (Malkov & Yashunin, 2020) or ScaNN (Guo et al., 2020)) when memory is constrained or when the dataset is especially large. In this work, we ask how to make an IVF index reach a target recall while minimizing the query time.

In IVF, there is one algorithmic knob for increasing recall at the cost of query time. A query finds its true nearest neighbor whenever the list containing that neighbor is scanned. So, to increase recall, the query algorithm can scan ℓ lists, usually those of the ℓ nearest centroids, where ℓ is a chosen parameter. Every query now pays for the size of the ℓ scanned lists, irrespective of whether the query needed those extra lists. Furthermore, in high dimensions many query-neighbor pairs straddle cluster boundaries; the relevant boundary is unknown at query time, and covering all of them degrades to a linear scan (the "curse of dimensionality" (Andoni et al., 2018)).[1]

[1] Covering every cluster that may hold a query's nearest neighbor is intractable, but we will see that samples from the query distribution identify a few clusters that cover most queries.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A natural remedy is to store dataset vectors in more than one list; the simplest rule stores each dataset vector in its closest r lists. Beyond that, some other rules have been explored: In SPANN (Chen et al., 2021), a vector is stored in every list whose centroid is a $(1+\epsilon_1)$-approximate closest centroid.[2] Another method, SOAR (Sun et al., 2023), replicates dataset vectors to an additional centroid so that residual vectors are nearly orthogonal. These rules decide where to store from the dataset alone, and this causes two failures:

1. A query-unaware method may duplicate dataset vectors which no query would use. This has a doubly negative effect: it increases the size of the index without improving recall, and it makes other queries slower by lengthening many lists.
2. More fundamentally, query-unaware methods can be particularly lossy in cross-modal retrieval (e.g., text queries search image embeddings), also known as "out-of-distribution" queries (Simhadri et al., 2024; Jääsaari et al., 2026). In general, queries may be routed to clusters whose centroids do not lie within the closest centroids of the nearest neighbor.

This work designs a query-aware method for dataset placement.[3] Given a sample of queries before one builds the index, one can compute "ground-truth" query-neighbor pairs and see exactly how an IVF index would perform on these pairs. On a query-neighbor pair (q, p), the query q (which aims to find p) scans the ℓ lists of its nearest centroids, and its query time is the sum of the sizes of the scanned lists; the query succeeds whenever p was placed in one of these ℓ lists.

> **Main Idea**: Given a sample of query-neighbor pairs, find an allocation (i.e., a placement of dataset points to clusters) which minimizes the number of candidates when scanning ℓ lists while ensuring a target recall on the observed query-neighbor pairs.

The above problem can be modeled exactly as a partial set cover (equivalently, maximum coverage problem) (Nemhauser et al., 1978; Wolsey, 1982; Khuller et al., 1999). We solve the partial set cover problem using a cost-benefit greedy algorithm, which has worst-case approximation guarantees and produces the entire recall–cost tradeoff in a single run. We use the linear-programming relaxation to certify primal-dual bounds on how accurate the cost-benefit greedy algorithm is on the instances we ran (always within 1.3% of optimal, Table 6); this shows that, while partial set cover is NP-hard, the instances encountered are certifiably easy in practice. Furthermore, we provide generalization bounds showing how the optimal allocation found on the samples will generalize in expected query time and recall to draws from the same distribution (Section 3.3).

Our approach sits squarely within the data-driven algorithm design paradigm of Gupta & Roughgarden (2017), as does Khanna et al. (2026) in the context of nearest-neighbor search. The IVF index is tuned on a "ground-truth" set of samples from the query distribution and judged on that same distribution. We address two potential concerns:

- **The query-aware method needs samples from the query distribution.** An IVF index is built precisely because it will serve a long stream of future queries, so the "ground-truth" samples could be collected on the fly (for example, by maintaining a small fraction of the queries).
- **The performance guarantees are with respect to a query distribution.** The field already makes this assumption implicitly; every benchmark evaluates on held-out queries, and every index has hyperparameters which are tuned using queries. Distribution shift may be addressed by maintaining some fraction of queries on the fly and rebuilding the index every so often.

On the 24 datasets of the VIBE benchmark (Jääsaari et al., 2026), at recall@100 $= 0.9$, query-aware IVF is faster than both vanilla IVF and query-agnostic replication on every dataset, by a

[2] The parameter $\epsilon_1 > 0$ is a hyperparameter to be chosen. The term "query-aware *pruning*" in Chen et al. (2021) refers to a different notion than query-aware *allocation* in this work. There, query-aware pruning scans the $(1+\epsilon_2)$-approximate closest centroids instead of the ℓ closest centroids, where $\epsilon_2 > 0$ is another hyperparameter.

[3] To the best of our knowledge, while query-aware methods have been used to learn rankings of clusters (Vecchiato et al., 2024) and stopping rules (Li et al., 2020), this is the first work to learn dataset placement (i.e., allocations to clusters) using the query distribution.

median $1.63\times$ over the faster of the two. The gains are largest when queries are out of distribution (a median $3.88\times$ over the nine such datasets) and up to $21.9\times$. The price of these speedups over vanilla IVF is paid in memory and build time. The index stores a median 2.58 entries per vector; while this can be expensive when storing vectors explicitly (for streaming reads), it can be mitigated by quantization (Jégou et al., 2011) and/or optimizing the memory layout. The build time is larger than that of vanilla IVF because the cost-benefit greedy algorithm must be run, but we emphasize that this cost may be incurred only periodically (to adapt to distribution shift). Against graph and quantization indices (Subramanya et al., 2019; Malkov & Yashunin, 2020; Sun et al., 2023; Chen et al., 2024), most of which are faster in RAM on most datasets, query-aware IVF closes much of the gap, and it is the fastest of all methods in that comparison on the two out-of-distribution datasets where HNSW and DiskANN never reach recall 0.9.

Summary of Contributions and Organization. After background in Section 2, we contribute the following.

- **Formulation.** With the centroids and probe count fixed, we cast the dataset allocation as a partial set cover problem over a sample of query-neighbor pairs. The cost model exactly minimizes the number of scanned candidates subject to a recall constraint (Section 3).
- **Algorithm and Guarantees.** A cost-benefit greedy, run with an indexed heap in time near-linear in the instance, produces the whole recall-vs-time frontier in one pass (Section 3.2). Linear programming certifies that the solutions found are near-optimal on every dataset (Appendix C). We provide a sample-complexity bound which can provide query time guarantees for the learned index on the query distribution (Section 3.3).
- **Experiments.** Our experiments are meant to answer the question: does query-aware dataset allocation provide query time improvements at a particular recall over vanilla IVF and natural query-unaware methods? On all 24 VIBE datasets, query-aware IVF dominates vanilla IVF and query-agnostic replication, with the largest gains on out-of-distribution queries (Section 4). Compared with graph indices (which are often faster than IVF), query-aware IVF can narrow some of these gaps, and even outperform graph indices on two datasets (Appendix D).

Additional Related Work on Out-of-Distribution Datasets. In many applications the queries follow a different distribution from the indexed data, most prominently in cross-modal retrieval, where text queries search a base set of image embeddings; this out-of-distribution (OOD) setting was a dedicated track of the NeurIPS'23 Big-ANN competition (Simhadri et al., 2024) and is a focus of the VIBE benchmark (Jääsaari et al., 2026). Graph indices built for in-distribution queries degrade on OOD queries, and the leading remedies use a sample of queries at build time: OOD-DiskANN (Jaiswal et al., 2022) feeds a small query sample into graph construction, and RoarGraph (Chen et al., 2024) builds a bipartite graph between queries and their nearest base points and projects it onto the base set. For clustering- and partition-based indices, queries have been used to learn the partition itself (Dong et al., 2020; Gupta et al., 2022), which lists to probe (Vecchiato et al., 2024), and when to stop probing (Li et al., 2020). To our knowledge, none of these uses a query sample to decide which lists each vector is stored in, which is the question we study.

2 Preliminaries

Nearest Neighbor Search. We are given a *base set* of vectors $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ with a dissimilarity function $\delta(q, p)$, which is usually either the Euclidean distance $\|q - p\|$ or the negative inner product $-\langle q, p\rangle$. For a query $q \in \mathbb{R}^d$, we write $G_K(q) \subseteq P$ for the K base vectors nearest to q under δ. An *approximate nearest neighbor* (ANN) algorithm preprocesses P into an *index*. Then, given as input a query q, it returns K base vectors $\hat{G}(q)$. The quality of the index is governed by its recall@K, the average of $|\hat{G}(q) \cap G_K(q)|/K$ over queries q.

2.1 IVF Indices

Vanilla IVF. IVF (Sivic & Zisserman, 2003; Jégou et al., 2011) is a type of ANN index built in the following two steps. First, k-means (Lloyd, 1982), typically run on P, gives t centroids $\mu_1, \ldots, \mu_t$.

Second, each base vector p goes into the list of its nearest centroid, its *primary list*:

$$a(p) = \arg\min_{j \in [t]} \|p - \mu_j\|^2, \qquad L_j = \{p \in P : a(p) = j\}.$$

A query q ranks the centroids by δ. Given a *probe count* parameter $\ell \geq 1$, the top ℓ ranked centroids form a *probe set* $R_\ell(q) \subseteq [t]$. For each $j \in R_\ell(q)$ and $p \in L_j$, the index computes $\delta(q, p)$ and returns the K nearest, $\hat{G}(q)$. A point $p \in G_K(q)$ is returned if and only if $a(p) \in R_\ell(q)$.

Replication. One strategy to improve the recall of an IVF index is to store each vector in multiple lists. A *placement* $A \subseteq P \times [t]$ is a set of assignments where $(p, j) \in A$ indicates that $p \in L_j$. The simplest heuristic, which we call *query-agnostic IVF*, is given by placing each point in the lists of its r nearest centroids for a parameter $r \geq 1$:

$$A_r = \{(p, j) : p \in P,\ \mu_j \text{ is among the } r \text{ centroids minimizing } \|p - \mu_j\|\}. \tag{1}$$

The placement A_1 recovers vanilla IVF. We call this strategy query-agnostic because it is computed from P and the centroids alone; it is independent of Q.

Observation 1. $p \in G_K(q)$ is returned under A if and only if $(p, j) \in A$ for some $j \in R_\ell(q)$.

3 Query-Aware IVF

We present a query-aware IVF approach that uses the learning set to optimize placements. We will consider a fixed set of centroids and probe count ℓ, so that the probe set $R_\ell(q)$ is fixed and the only remaining choice is the placement strategy. We retain the *primary placement* $F = \{(p, a(p)) : p \in P\}$ of vanilla IVF and use Q to choose copies (p, j) to append.

Let $\pi_j = \Pr_{q \sim \mathcal{Q}}[j \in R_\ell(q)]$ be the *probe frequency* of list j. By Observation 1, we can write the recall of A and the expected number of candidates (for queries $q \sim \mathcal{Q}$) as follows:

$$\mathrm{Recall}(A) = \mathbb{E}_{q \sim \mathcal{Q}}\Big[\tfrac{1}{K}\,\big|\{p \in G_K(q) : \exists\, j \in R_\ell(q),\ (p, j) \in A\}\big|\Big], \tag{2}$$

$$\mathrm{Cost}(A) = \mathbb{E}_{q \sim \mathcal{Q}}\Big[\sum_{j \in R_\ell(q)} |L_j(A)|\Big] = \mathbb{E}_{q \sim \mathcal{Q}}\Big[\sum_{(p,j) \in A} \mathbf{1}[j \in R_\ell(q)]\Big] = \sum_{(p,j) \in A} \pi_j, \tag{3}$$

where the last equality follows from linearity of expectation.[4] Adding an entry (p, j) to A thus costs π_j candidates per query and raises the recall by the probability that the query newly reaches p by probing j. For a bound C on the number of candidates per query, an optimal placement solves

$$\max_A \ \mathrm{Recall}(A) \quad \text{subject to} \quad \mathrm{Cost}(A) \leq C. \tag{4}$$

3.1 Learning the placement as partial set cover

We simplify equation 4 in two steps. First, since $\mathcal{Q}$ is unknown, we replace it by the learning set Q, a sample of m queries drawn independently from $\mathcal{Q}$ ($Q \sim \mathcal{Q}^m$): the *training* recall and cost $\widehat{\mathrm{Recall}}$ and $\widehat{\mathrm{Cost}}$ average over $q \in Q$ instead of taking the expectation over $\mathcal{Q}$, and π_j becomes $\hat{\pi}_j = |\{q \in Q : j \in R_\ell(q)\}|/m$. For each $q \in Q$ we compute $R_\ell(q)$ and the exact neighbors $G_K(q)$. Second, we add copies only for the query–neighbor pairs vanilla IVF misses, the *demand pairs*:

$$\mathcal{D} = \big\{(q, p) : q \in Q,\ p \in G_K(q),\ a(p) \notin R_\ell(q)\big\}. \tag{5}$$

A copy of p in list j or a placement entry (p, j) *covers* $(q, p) \in \mathcal{D}$ if $j \in R_\ell(q)$ and p is in list j. We further define that the *eligible copies* are $\mathcal{C} = \{(p, j) : (q, p) \in \mathcal{D},\ j \in R_\ell(q)\}$.

The eligible copy (p, j) covers the demand pairs $\mathcal{D}_{pj} = \{(q, p) \in \mathcal{D} : j \in R_\ell(q)\}$. For $S \subseteq \mathcal{C}$, let

$$f(S) = \Big|\bigcup_{(p,j) \in S} \mathcal{D}_{pj}\Big|, \qquad \hat{\pi}(S) = \sum_{(p,j) \in S} \hat{\pi}_j$$

[4] One unit per candidate is the cost model used in our experiments. Any per-entry cost $w_{p,j} \geq 0$, such as pages read from disk, fits the formulation and leaves the algorithm unchanged.

Algorithm 1 Building a query-aware placement at probe count ℓ

Require: base set P, centroids $\mu_1, \ldots, \mu_t$, learning set Q, neighbor count K, probe count ℓ
Ensure: copies $c_1, \ldots, c_T$ in the order added, so $A(s)$ is as in equation 9

```
compute a(p), R_ℓ(q), G_K(q) and π̂_j; build D, C and every D_pj
Δ(p, j) ← |D_pj| for every (p, j) ∈ C                    ▷ Δ(c) = Δ(c | S) throughout
indexed max-heap H over C keyed by Δ(p, j)/π̂_j; uncovered pairs U ← D; sequence ← [ ]
while U ≠ ∅ do
    (p, j) ← extract the top of H; append (p, j) to the sequence
    for each (q, p) ∈ D_pj ∩ U do
        remove (q, p) from U; for j' ∈ R_ℓ(q) \ {j}, decrement Δ(p, j') and its key
return the sequence
```

be the number of demand pairs S covers and its cost. Since no eligible copy lies in F,

$$\widehat{\mathrm{Recall}}(F \cup S) = \widehat{\mathrm{Recall}}(F) + \frac{f(S)}{mK}, \qquad \widehat{\mathrm{Cost}}(F \cup S) = \widehat{\mathrm{Cost}}(F) + \hat{\pi}(S),$$

so equation 4 on Q, over the indices $A = F \cup S$ and with $C = \widehat{\mathrm{Cost}}(F) + B$ for a budget of B extra candidates per query, becomes

$$\max_{S \subseteq \mathcal{C}} f(S) \quad \text{subject to} \quad \hat{\pi}(S) \le B. \tag{6}$$

With demand pairs as elements and each copy as the set of pairs it covers, weighted $\hat{\pi}_j$, this is budgeted maximum coverage (Nemhauser et al., 1978; Khuller et al., 1999). Since it is more intuitive for a user to input a target recall rather than a budget, we instead solve its covering version, a partial set cover problem (Wolsey, 1982). In our notation, since $\widehat{\mathrm{Recall}}(F) = 1 - |\mathcal{D}|/(mK)$, reaching a training recall $\widehat{\mathrm{Recall}}(F \cup S) \ge \alpha$ means covering $k_\alpha = \lceil |\mathcal{D}| - (1-\alpha)mK \rceil$ demand pairs, and the problem is to find the cheapest such S:

$$\min_{S \subseteq \mathcal{C}} \hat{\pi}(S) \quad \text{subject to} \quad f(S) \ge k_\alpha. \tag{7}$$

All quantities in the above problem (R_ℓ, $\mathcal{D}$ and $\mathcal{C}$) depend on the probe count ℓ, so we solve one instance per probe count. Problem (7), a partial set cover problem, is NP-hard, so we approximate it with a greedy algorithm in the next subsection.

3.2 The cost-benefit greedy algorithm

We approximate equation 7 with the cost-benefit greedy Algorithm 1. Starting from $S = \emptyset$, it repeatedly adds the eligible copy with the largest gain per unit of cost,

$$c^\star = \operatorname*{arg\,max}_{c=(p,j) \in \mathcal{C} \setminus S} \frac{\Delta(c \mid S)}{\hat{\pi}_j}, \qquad \Delta(c \mid S) = f(S \cup \{c\}) - f(S), \tag{8}$$

until all demand is covered. Gains only fall: covering a pair removes one unit of gain from each of its other $\ell - 1$ covering copies. Since the greedy only adds copies, the order $c_1, \ldots, c_T$ in which it adds them gives a nested family of indices

$$A(s) = F \cup S_s, \qquad S_s = \{c_1, \ldots, c_s\}, \qquad 0 \le s \le T, \tag{9}$$

from vanilla IVF at $s = 0$ to full coverage at $s = T$, so one run traces the whole recall–cost trade-off at probe count ℓ. The training recall of $A(s)$ is

$$\widehat{\mathrm{Recall}}(A(s)) = \frac{mK - |\mathcal{D}| + f(S_s)}{mK}, \tag{10}$$

and cutting at the first s where it reaches α gives our solution of equation 7; Section 4 builds its indices this way. An index built at probe count ℓ can be searched at any $\ell' \ge \ell$. The algorithm keeps $\mathcal{C}$ in an indexed max-heap keyed by $\Delta(c \mid S)/\hat{\pi}_j$ and lowers keys in place, so each step extracts the exact maximizer of equation 8, in $O\big((|\mathcal{C}| + \ell\,|\mathcal{D}|)\log|\mathcal{C}|\big)$ time and $O(|\mathcal{C}| + \ell\,|\mathcal{D}|)$ memory.

3.3 GUARANTEES

We bound how far the greedy algorithm is from the optimum on Q, then how its index generalizes to $\mathcal{Q}$.

How far from optimal is the greedy algorithm? The prefix cut at α is also a solution of equation 6 at the budget B it spends, and there it is within $1 - 1/e \approx 0.63$ of the optimum (Khuller et al., 1999). On our instances it does far better: against the exact optimum of the linear relaxation at the same budget, which bounds every placement from above, the greedy behind each dataset's query-aware index at recall 0.90 reaches a median 0.995 of it and never less than 0.987 (Appendix C).

Generalization. The greedy algorithm optimizes on Q, but the index serves $\mathcal{Q}$. We bound its query time and recall there, with probability over the draw $Q \sim \mathcal{Q}^m$; as in equation 3, $\widehat{\mathrm{Cost}}(A) = \sum_j \hat{\pi}_j |L_j(A)|$. Proofs are in Appendix A. For simplicity, both theorems assume that the centroids do not depend on Q, as when k-means runs on the base set.

Theorem 3.1 (Certified query time)**.** *Suppose the centroids do not depend on $Q \sim \mathcal{Q}^m$. Fix $\eta \in (0,1)$, and let t be the number of lists. With probability at least $1-\eta$, every placement A with $m\,\widehat{\mathrm{Cost}}(A) \geq 2|A|\log(2t/\eta)$ satisfies $\mathrm{Cost}(A) \leq (1+\varepsilon)\,\widehat{\mathrm{Cost}}(A)$, where $\varepsilon = 2\sqrt{2|A|\log(2t/\eta)/(m\,\widehat{\mathrm{Cost}}(A))}$.*

If every list holds $|A|/t$ entries, then $\widehat{\mathrm{Cost}}(A) = \ell|A|/t$, since every query probes ℓ lists and so $\sum_j \hat{\pi}_j = \ell$; the condition then reads $m\ell/t \geq 2\log(2t/\eta)$, and

$$\varepsilon = 2\sqrt{\frac{2t\log(2t/\eta)}{m\ell}}. \tag{11}$$

The error ε is thus small once $m\ell/t$, the average number of learning queries probing a list, is much larger than $\log(2t/\eta)$, whatever the size of the index.

Theorem 3.2 (Certified recall for sparse placements)**.** *Let $\sigma(A) = 1 - \mathrm{Recall}(A)$ and $\hat{\sigma}(A) = 1 - \widehat{\mathrm{Recall}}(A)$. Suppose the centroids do not depend on $Q \sim \mathcal{Q}^m$. Fix $\eta \in (0,1)$, and for $b \geq 0$ put*

$$\Lambda = b\log(nt) + \log(nt/\eta) \;\approx\; b\log(nt). \tag{12}$$

With probability at least $1-\eta$, for every b, every index $A = F \cup S$ with b copies ($S \cap F = \emptyset$, $|S| = b$) satisfies

$$\sigma(A) \;\leq\; \hat{\sigma}(A) + \sqrt{\frac{2\hat{\sigma}(A)\Lambda}{m}} + \frac{2\Lambda}{m}. \tag{13}$$

The bound says that the learning set needs to grow with the number b of copies bought, but only logarithmically with the size n of the dataset.

4 EXPERIMENTS

We compare the empirical performance of query-aware IVF with that of vanilla IVF and query-agnostic IVF. Our goal is to measure the purely algorithmic improvement that comes from learning where to store each vector. Optimizations orthogonal to the placement, such as quantization, storage layout and disk I/O, are not a main focus in this section.

Because the gain of query-aware IVF is purely algorithmic, it composes with such optimizations: although we did not optimize the combination, directly pairing query-aware IVF with the RaBitQ quantizer (Gao & Long, 2024) makes it faster on 16 of the 24 datasets, by a median $2.7\times$ at recall 0.9. We also compare our method to other popular ANN indices. See Appendix D for details.

4.1 SETUP

Datasets. We evaluate the methods on 24 datasets in VIBE (Jääsaari et al., 2026), a benchmark of text and image embeddings. Most of the datasets consist of around 1 million vectors in several

hundred dimensions, and each comes with a metric (either Euclidean, cosine, or inner product). 15 of the datasets have *in-distribution* (ID) queries, drawn from the same embedding distribution as the dataset. The other 9 have *out-of-distribution* (OOD) queries, drawn from a different encoder or a different modality (e.g., text queries for image embeddings). In addition to the base vectors, each dataset comes with 1000 test queries with their exact 100 nearest neighbors (in the given metric). For cosine similarity, we normalize all vectors to unit length and use Euclidean distance, which ranks neighbors identically. For every metric, k-means is standard Euclidean k-means. For inner product, base vectors are assigned to their nearest centroids in Euclidean distance, while queries probe the lists whose centroids have the largest inner product with them, ranking candidates by inner product.

Learning queries. The query-aware method uses a sample from the query distribution at build time. The OOD datasets on VIBE are supplied with a learning set of between about 10^5 and 10^6 queries along with their 100 nearest neighbors (see Table 4). For the ID datasets, which have none, the learning set is a random 20% of the base vectors, which stay in the base set, with their 100 nearest neighbors. Appendix B.3 varies the size of the learning set.

Methods. The primary methods we compare are vanilla IVF, query-agnostic IVF, and query-aware IVF. In Appendix D, we additionally compare query-aware and vanilla IVF against five graph-based and quantization-based indices: HNSW (Malkov & Yashunin, 2020), DiskANN (Subramanya et al., 2019), ScaNN (Guo et al., 2020), SOAR (Sun et al., 2023), and RoarGraph (Chen et al., 2024). The first four are query-unaware; RoarGraph is query-aware.

Build-time parameters. By default, IVF has one build-time parameter: t = the number of centroids used to define the clustering. We use the following parameter settings:

- For all methods, we use $t = \lfloor \sqrt{n} \rfloor$ centroids. For ID datasets, k-means is fit on base vectors. For OOD datasets, a second option is to fit on the learning set, which query-aware IVF always uses; the other methods sweep both options.
- Query-agnostic IVF has the additional parameter r, the number of copies created for each point. We sweep over all $r \in \{1, 2, 4, 8, 16, 32\}$.
- Query-aware IVF has two parameters. The first is α_{train}, the recall threshold it uses during training. The second is ℓ_{train}, the probe count it optimizes for during training. We sweep over $\alpha_{\text{train}} \in \{0.85, 0.9, 0.95, 0.99\}$ and $\ell_{\text{train}} \in \{1, 2, 4\}$.

Query-time parameters. All methods have a single query-time parameter: ℓ = the number of clusters probed during query time. For each of the above build-time configurations, we sweep ℓ over powers of 2 beginning at 1. For vanilla IVF and query-agnostic IVF we cap at $\ell = 256$. For query-aware IVF, we sweep ℓ between ℓ_{train} and 64. We terminate a sweep when the recall@100 reaches 0.995 or the number of candidates scanned exceeds n.

Metrics and hardware. We report recall@100 on all test queries, defined as the average proportion of a query's top 100 nearest neighbors found during search. We measure latency on the first 200 test queries, after a 50 query warm-up phase; each query is timed three times and we report the median of the per-query means. All experiments were run on a dual-socket AMD EPYC 9575F machine (2×64 cores, 256 threads, AVX-512) with 3TB of RAM, using a customized version of FAISS 1.15.0 (Douze et al., 2026) to support the replication variants. Builds used 8 threads while all reported latencies were single-threaded measurements. We dropped any build whose index would exceed 512GB, which excluded five query-agnostic and two query-aware configurations.

4.2 Results

Recall against latency. Table 1 reports the single-thread QPS of each method at recall@100 = 0.9, and Figure 1 the full recall–QPS tradeoff. On all 24 datasets query-aware IVF beats both baselines, by at least 1.31× and by a median 1.63×. The speedup is greatest on the nine OOD datasets, a median 3.88× and 14–22× on three of them, against a median 1.41× on the fifteen ID datasets. Counted in scanned candidates, which do not depend on the machine, the improvements are nearly the same, so they are algorithmic (Appendix B.1).

Index size and build time. Table 2 reports the storage space of the query-aware method. At recall@100 $= 0.9$, the query-aware index stores a median 2.58 copies per vector, counting the original vector, and a maximum of 21.4. The index size grows by nearly the same factor. Within a single index, the copies are spread unevenly: against the median of 2.58, the *most replicated* vector appears in a median 35 clusters. Table 2 also reports the build time of the query-aware method. The median increase with respect to IVF is $2.17\times$, but it ranges from $0.83\times$ (due to k-means being faster on the learning set than on the base vectors) up to $36.7\times$. The build time measurements do not include the time of computing the 100 nearest neighbor ground truth of each learning set query.

Table 1: Highest single-thread QPS achieving recall@100 $= 0.9$, over each method's parameter sweep. Speed-up is over the faster baseline; dashes indicate that the target recall was not reached, which for vanilla IVF on three datasets is because its sweep ends at $\ell = 256$.

				QPS at recall@100 $= 0.9$			
dataset	queries	n	d	IVF	agnostic	query-aware	speed-up
coco-nomic	OOD	282,360	768	232	234	3,385	**14.48**×
cqadupstack-lemur	OOD	457,149	2048	—	26.5	115	**4.35**×
cqadupstack-muvera	OOD	457,149	5120	10.6	11.6	21.5	**1.86**×
hotpotqa-harrier	OOD	5,233,329	640	52.2	52.2	95.2	**1.82**×
imagenet-align	OOD	1,281,167	640	260	900	2,114	**2.35**×
laion-clip	OOD	1,000,448	512	—	21.9	480	**21.90**×
llama	OOD	256,921	128	1,067	1,118	17,236	**15.42**×
yandex	OOD	1,000,000	200	789	789	1,412	**1.79**×
yi	OOD	187,843	128	2,429	2,429	9,414	**3.88**×
agnews-mxbai	ID	769,382	1024	631	631	1,082	**1.71**×
arxiv-nomic	ID	1,344,643	768	570	600	937	**1.56**×
ccnews-nomic	ID	495,328	768	1,008	1,008	1,382	**1.37**×
celeba-resnet	ID	201,599	2048	573	573	863	**1.51**×
codesearchnet-jina	ID	1,374,067	768	326	504	678	**1.34**×
dpr-jina	ID	20,969,760	768	98.2	98.2	128	**1.31**×
glove	ID	1,192,514	200	—	128	211	**1.65**×
gooaq-distilroberta	ID	1,475,024	768	542	542	877	**1.62**×
imagenet-clip	ID	1,281,167	512	1,451	1,621	2,256	**1.39**×
inaturalist-resnet	ID	499,000	2048	400	400	835	**2.09**×
landmark-dino	ID	760,757	768	830	861	1,320	**1.53**×
landmark-nomic	ID	760,757	768	1,397	1,503	2,121	**1.41**×
msmarco-qwen	ID	8,840,823	1024	82.2	130	176	**1.35**×
simplewiki-openai	ID	260,372	3072	338	338	454	**1.34**×
yahoo-minilm	ID	677,305	384	835	1,438	2,027	**1.41**×
median							**1.63**×

Table 2: Index size and build time of the configurations in Table 1.

	entries	index size (GB)			build time (s)		
dataset	per vector	IVF	query-aware	increase	IVF	query-aware	increase
coco-nomic	7.45	0.87	6.48	**7.44**×	1.54	3.29	**2.13**×
cqadupstack-lemur	10.04	3.75	37.63	**10.02**×	6.22	83.9	**13.50**×
cqadupstack-muvera	21.41	9.38	200.55	**21.38**×	15.0	134	**8.92**×
hotpotqa-harrier	1.70	13.45	22.87	**1.70**×	25.6	21.3	**0.83**×
imagenet-align	2.43	3.29	8.01	**2.43**×	5.74	10.5	**1.83**×
laion-clip	9.74	2.06	20.04	**9.73**×	3.47	69.1	**19.95**×
llama	1.23	0.13	0.16	**1.23**×	0.37	3.66	**9.84**×
yandex	8.05	0.81	6.50	**8.04**×	1.52	55.6	**36.68**×
yi	3.92	0.10	0.38	**3.91**×	0.25	1.62	**6.57**×
agnews-mxbai	2.27	3.16	7.16	**2.27**×	5.21	9.14	**1.75**×
arxiv-nomic	2.48	4.15	10.28	**2.48**×	7.19	15.8	**2.19**×
ccnews-nomic	2.45	1.53	3.73	**2.44**×	2.65	5.13	**1.93**×
celeba-resnet	3.24	1.66	5.35	**3.23**×	3.48	4.38	**1.26**×
codesearchnet-jina	2.16	4.24	9.16	**2.16**×	7.24	19.7	**2.71**×
dpr-jina	2.75	64.60	177.83	**2.75**×	150	459	**3.06**×
glove	5.61	0.96	5.40	**5.60**×	2.16	28.0	**12.96**×
gooaq-distilroberta	2.56	4.55	11.65	**2.56**×	10.3	21.1	**2.05**×
imagenet-clip	1.29	2.64	3.40	**1.29**×	4.65	9.28	**1.99**×
inaturalist-resnet	1.87	4.10	7.66	**1.87**×	6.61	8.84	**1.34**×
landmark-dino	2.60	2.35	6.10	**2.60**×	3.86	8.26	**2.14**×
landmark-nomic	2.88	2.35	6.74	**2.87**×	3.87	6.13	**1.58**×
msmarco-qwen	1.64	36.29	59.64	**1.64**×	72.9	163	**2.24**×
simplewiki-openai	3.47	3.21	11.11	**3.46**×	5.37	6.38	**1.19**×
yahoo-minilm	1.87	1.05	1.96	**1.87**×	1.92	6.09	**3.17**×
median	2.58			**2.58**×			**2.17**×

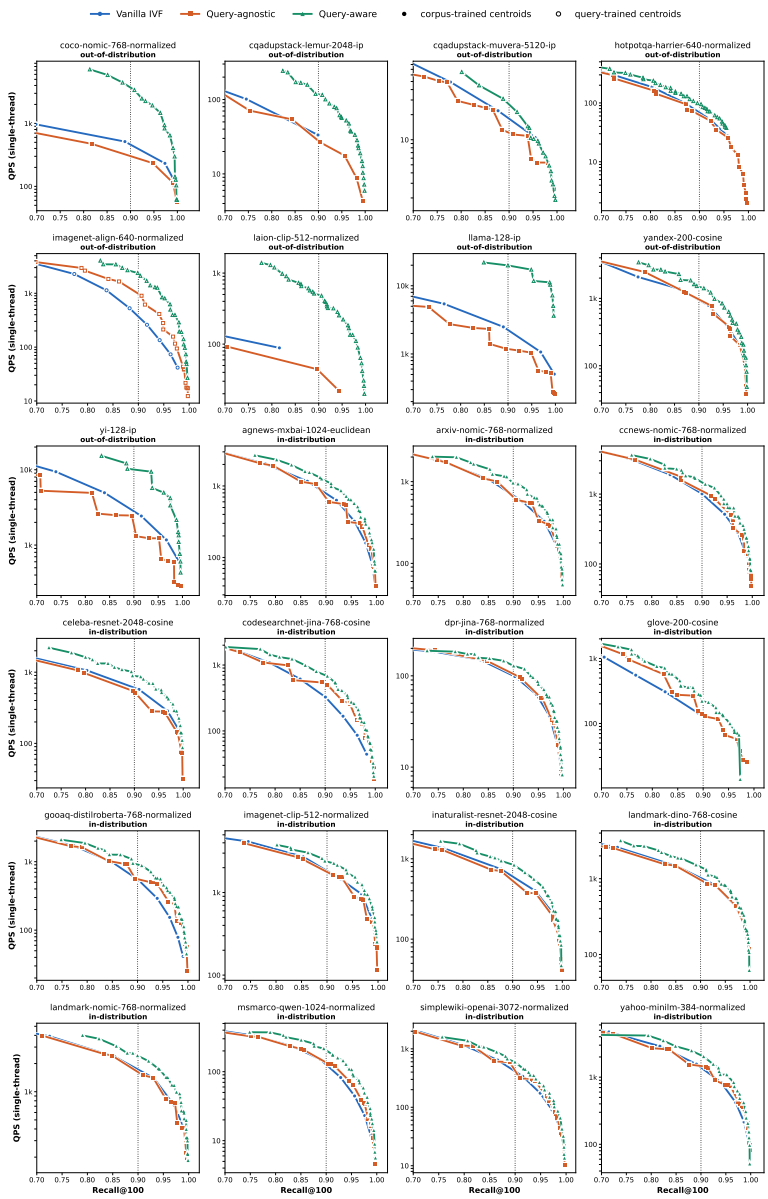


Figure 1: Single-thread QPS against recall@100, each method's fastest configuration per recall. Filled/hollow markers: centroids fit on the base vectors/learning set. Query-aware builds target $\alpha_{\text{train}} \geq 0.85$, so their curves start higher. Dotted: recall 0.9.

4.3 Discussion

We believe our experimental results demonstrate the effectiveness of query-aware IVF, particularly when queries come from outside the base distribution. Its main costs are index size and build time. For index size, one could store a pointer to each point rather than a copy; this saves space at the cost of query time, since lists would no longer be scanned contiguously; we did not try it.

Limitations. Query-aware IVF needs learning queries together with their ground truth nearest neighbors, so they must be collected and their neighbors computed. In practice, the learning queries can be collected from previously served queries, and the cost of computing their ground truth is amortized over the many queries the index then serves.

AI USE STATEMENT

In this work, we used generative AI tools (Claude Code) to help implement the method and the experiments. We have not used them to generate synthetic data; translation, dataset cleaning and qualitative data analysis are not applicable to this work. We also used them to draft and edit the paper and to prepare its figures, tables and references. We have reviewed all AI-assisted work and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithm 1 specify the method, and Appendix A proves the guarantees of Section 3.3. All 24 datasets come from the public VIBE benchmark (Jääsaari et al., 2026); Section 4.1 describes how we form the learning sets and gives every build-time and query-time parameter we sweep, the timing protocol and the hardware, and Appendix B adds the sizes of the learning sets. Our implementation extends FAISS (Douze et al., 2026). We would be glad to release the code, together with the scripts that reproduce every table and figure, upon acceptance of the paper.

REFERENCES

Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 2018.

Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. RoarGraph: A projected bipartite graph for efficient cross-modal approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 17(11):2735–2749, 2024.

Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: Highly-efficient billion-scale approximate nearest neighbor search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations (ICLR)*, 2020.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The Faiss library. *IEEE Transactions on Big Data*, 12(2):346–361, 2026.

Jianyang Gao and Cheng Long. RaBitQ: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 2(3), 2024.

Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. Practical and asymptotically optimal quantization of high-dimensional vectors in Euclidean space for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 3(3), 2025.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning (ICML)*, pp. 3887–3896, 2020.

Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. BLISS: A billion scale index using iterative re-partitioning. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 486–495, 2022.

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

Qi Huangfu and J. A. Julian Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

Elias Jääsaari, Ville Hyvönen, Matteo Ceccarello, Teemu Roos, and Martin Aumüller. VIBE: Vector index benchmark for embeddings. *Journal of Data-centric Machine Learning Research*, 2026.

Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. OOD-DiskANN: Efficient and scalable graph ANNS for out-of-distribution queries. *arXiv preprint arXiv:2211.12850*, 2022.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

Sanjeev Khanna, Ashwin Padaki, and Erik Waingarten. Learning partition trees for nearest neighbor search. *arXiv preprint arXiv:2607.09909*, 2026.

Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In *ACM SIGMOD International Conference on Management of Data*, pp. 2539–2554, 2020.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982.

Yu A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Zihao Wan, Jie Yin, and Ben Huang. Results of the big ann: Neurips'23 competition. *arXiv preprint arXiv:2409.17424*, 2024. doi: 10.48550/arXiv.2409.17424.

Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 1470–1477, 2003.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. SOAR: Improved indexing for approximate nearest neighbor search. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3189–3204, 2023.

Nitish Upreti, Harsha Vardhan Simhadri, Hari Sudan Sundar, et al. Cost-effective, low latency vector search with Azure Cosmos DB. *arXiv preprint arXiv:2505.05885*, 2025.

Thomas Vecchiato, Claudio Lucchese, Franco Maria Nardini, and Sebastian Bruch. A learning-to-rank formulation of clustering-based approximate nearest neighbor search. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2261–2265, 2024.

Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, pp. 2614–2627, 2021.

Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

Alexandros Zeakis, George Papadakis, Dimitrios Skoutas, and Manolis Koubarakis. Pre-trained embeddings for entity resolution: an experimental analysis. *Proceedings of the VLDB Endowment*, 16(9):2225–2238, 2023.

A Proofs

Both proofs use the lower tail of the multiplicative Chernoff bound: if $\bar{Z}$ is the average of m independent $[0,1]$-valued variables with mean μ, then $\Pr[\,\bar{Z} \le \mu - w\,] \le \exp(-mw^2/(2\mu))$ for $w \ge 0$. Taking $w = \sqrt{2\mu\lambda/m}$, with probability at least $1 - e^{-\lambda}$,

$$\mu \le \bar{Z} + \sqrt{2\mu\lambda/m} \quad\Longrightarrow\quad \sqrt{\mu} \le \sqrt{\tfrac{\lambda}{2m}} + \sqrt{\tfrac{\lambda}{2m} + \bar{Z}} \quad\Longrightarrow\quad \mu \le \bar{Z} + \sqrt{\tfrac{2\bar{Z}\lambda}{m}} + \tfrac{2\lambda}{m}, \quad (14)$$

where the middle step solves the quadratic inequality in $\sqrt{\mu}$, and the last squares it, $\mu \le \bar{Z} + \lambda/m + \sqrt{\lambda^2/m^2 + 2\bar{Z}\lambda/m}$, and uses $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$.

Proof of Theorem 3.1. Write $\lambda = \log(2t/\eta)$ and $\hat{C} = \widehat{\mathrm{Cost}}(A)$. Fix a list j. The indicator that a query $q \sim \mathcal{Q}$ probes j has mean π_j, and since the centroids, and hence the probe sets, do not depend on Q, $\hat{\pi}_j$ is the average of m independent copies of it. By equation 14, $\pi_j \le \hat{\pi}_j + \sqrt{2\hat{\pi}_j\lambda/m} + 2\lambda/m$ fails with probability at most $e^{-\lambda} = \eta/(2t)$, so by a union bound over the t lists it holds for every j with probability at least $1 - \eta$. On that event, multiplying the bound for list j by $|L_j(A)| \ge 0$ and summing over j gives, since $\mathrm{Cost}(A) = \sum_j \pi_j |L_j(A)|$ by equation 3,

$$\mathrm{Cost}(A) \;\le\; \sum_j \Big(\hat{\pi}_j + \sqrt{\tfrac{2\hat{\pi}_j\lambda}{m}} + \tfrac{2\lambda}{m}\Big)|L_j(A)|.$$

Since $\sum_j |L_j(A)| = |A|$ and $\sum_j \hat{\pi}_j |L_j(A)| = \hat{C}$, the first and last terms on the right are $\hat{C}$ and $2\lambda|A|/m$. For the middle one, Cauchy–Schwarz, $\sum_j a_j b_j \le (\sum_j a_j^2)^{1/2} (\sum_j b_j^2)^{1/2}$ with $a_j = \sqrt{|L_j(A)|}$ and $b_j = \sqrt{\hat{\pi}_j |L_j(A)|}$, gives

$$\sum_j \sqrt{\tfrac{2\hat{\pi}_j\lambda}{m}}\,|L_j(A)| = \sqrt{\tfrac{2\lambda}{m}} \sum_j a_j b_j \le \sqrt{\tfrac{2\lambda}{m}}\,\sqrt{|A|\,\hat{C}}.$$

Hence $\mathrm{Cost}(A) \le \hat{C} + \sqrt{2\lambda|A|\hat{C}/m} + 2\lambda|A|/m$. With $x = 2\lambda|A|/(m\hat{C})$, which is at most 1 by assumption, so that $x \le \sqrt{x}$, this is $\hat{C}(1 + \sqrt{x} + x) \le \hat{C}(1 + 2\sqrt{x})$, which is the bound of Theorem 3.1. □

Proof of Theorem 3.2. Fix an index $A = F \cup S$ with $|S| = b$. The fraction of its K neighbors that a query $q \sim \mathcal{Q}$ misses under A lies in $[0,1]$ and has mean $\sigma(A)$, and since the centroids, and hence F and the probe sets, do not depend on Q, $\hat{\sigma}(A)$ is the average of m independent copies of it, so by equation 14 the bound equation 13 fails with probability at most $e^{-\lambda}$ for $\lambda = \Lambda$. There are at most $\binom{n(t-1)}{b} \le (nt)^b$ sets S of b copies, so a union bound over them fails with probability at most $(nt)^b e^{-\Lambda} = \eta/(nt)$, and a second union bound over the at most $n(t-1)+1 \le nt$ values of b gives total failure probability at most η. □

B Additional Experimental Details

B.1 Candidates scanned per query

Table 3 repeats Table 1 with the number of candidates scanned per query in place of QPS. Candidates are counted exactly and do not depend on the machine, its load or the implementation, yet they tell the same story: query-aware IVF scans fewer candidates than both baselines on all 24 datasets, by a

median $1.68\times$, against a median QPS speed-up of $1.63\times$. The speed-ups of Section 4 are therefore algorithmic: they come from scanning fewer candidates, not from engineering or hardware.

Table 3: Candidates scanned per query at recall@100 $= 0.9$, over each method's parameter sweep. Decrease is over the baseline with fewer candidates; dashes indicate that the target recall was not reached.

				candidates per query at recall@100 $= 0.9$			
dataset	queries	n	d	IVF	agnostic	query-aware	decrease
coco-nomic	OOD	282,360	768	59,605	59,605	4,633	**12.87**×
cqadupstack-lemur	OOD	457,149	2048	—	178,350	37,952	**4.70**×
cqadupstack-muvera	OOD	457,149	5120	170,586	139,489	77,679	**1.80**×
hotpotqa-harrier	OOD	5,233,329	640	298,807	298,807	160,971	**1.86**×
imagenet-align	OOD	1,281,167	640	66,263	27,640	7,473	**3.70**×
laion-clip	OOD	1,000,448	512	—	991,183	40,735	**24.33**×
llama	OOD	256,921	128	61,781	60,095	3,731	**16.11**×
yandex	OOD	1,000,000	200	63,452	63,452	34,694	**1.83**×
yi	OOD	187,843	128	27,453	27,453	8,828	**3.11**×
agnews-mxbai	ID	769,382	1024	16,518	16,244	9,275	**1.75**×
arxiv-nomic	ID	1,344,643	768	21,266	21,081	12,868	**1.64**×
ccnews-nomic	ID	495,328	768	13,402	13,393	8,197	**1.63**×
celeba-resnet	ID	201,599	2048	8,873	8,873	6,145	**1.44**×
codesearchnet-jina	ID	1,374,067	768	41,645	26,986	19,815	**1.36**×
dpr-jina	ID	20,969,760	768	82,391	82,391	55,608	**1.48**×
glove	ID	1,192,514	200	—	359,249	212,273	**1.69**×
gooaq-distilroberta	ID	1,475,024	768	23,324	23,324	13,910	**1.68**×
imagenet-clip	ID	1,281,167	512	11,132	11,132	7,486	**1.49**×
inaturalist-resnet	ID	499,000	2048	12,844	12,844	6,262	**2.05**×
landmark-dino	ID	760,757	768	15,710	15,692	10,047	**1.56**×
landmark-nomic	ID	760,757	768	8,730	8,730	5,981	**1.46**×
msmarco-qwen	ID	8,840,823	1024	106,727	60,052	42,332	**1.42**×
simplewiki-openai	ID	260,372	3072	10,209	10,177	7,543	**1.35**×
yahoo-minilm	ID	677,305	384	28,548	15,654	12,624	**1.24**×
median							**1.68**×

B.2 Learning queries

Table 4 gives the size of the learning set that each OOD dataset ships with; for the ID datasets, the learning set is a random 20% of the base vectors (Section 4.1).

Table 4: Number of learning queries for each OOD dataset in VIBE.

dataset	n	d	learning queries
coco-nomic	282,360	768	114,054
cqadupstack-lemur	457,149	2048	1,331,947
cqadupstack-muvera	457,149	5120	1,331,947
hotpotqa-harrier	5,233,329	640	96,845
imagenet-align	1,281,167	640	315,648
laion-clip	1,000,448	512	999,448
llama	256,921	128	255,921
yandex	1,000,000	200	1,000,000
yi	187,843	128	186,843

B.3 SIZE OF THE LEARNING SET

Table 5 varies the number of learning queries. On 20 of the 24 datasets (all but cqadupstack-lemur, cqadupstack-muvera, dpr-jina and msmarco-qwen), we build query-aware IVF from nested random subsets holding 10%, 20%, 50% and 100% of the learning set of Section 4.1. Centroids follow Section 4.1: on OOD datasets query-aware IVF uses centroids fit on the learning set, trained once on the whole learning set and shared by every subset, and vanilla IVF uses the better of those and centroids fit on the base vectors; on ID datasets every method uses centroids fit on the base vectors. This experiment differs from Section 4 in two ways: $\ell_{\text{train}} \in \{1, 2\}$, and we report candidates scanned per query rather than QPS, because these runs shared the machine with other jobs. For each subset, the table gives vanilla IVF's candidates per query at recall@$100 = 0.9$ divided by the fewest candidates of any query-aware configuration reaching that recall.

The median decrease grows from $1.37\times$ with 10% of the learning set to $1.45\times$, $1.66\times$ and $1.80\times$ with 20%, 50% and 100%. On every ID dataset the full learning set gives a larger decrease than 10% of it, and the ID median grows from $1.28\times$ to $1.69\times$. The OOD median grows faster, from $2.71\times$ to $10.27\times$: from 10% to 100% of the learning set the decrease grows $5.7\times$ on coco-nomic and $2.8\times$ on imagenet-align and laion-clip, while llama already reaches $19.6\times$ with 10%. Query-aware IVF scans fewer candidates than vanilla IVF everywhere except at 10% on hotpotqa-harrier ($0.67\times$) and codesearchnet-jina ($0.98\times$).

Table 5: Candidate decrease of query-aware IVF over vanilla IVF at recall@$100 = 0.9$, when learning from a fraction of the learning set of m queries. On OOD datasets query-aware IVF uses centroids fit on the learning set and vanilla IVF the better of those and centroids fit on the base vectors; on ID datasets both use centroids fit on the base vectors.

				fraction of the learning set			
dataset	queries	m	IVF candidates	10%	20%	50%	100%
coco-nomic	OOD	114,054	59,074	2.54×	6.23×	9.70×	14.40×
hotpotqa-harrier	OOD	96,845	296,520	0.67×	1.24×	1.11×	1.69×
imagenet-align	OOD	315,648	65,132	3.71×	6.27×	8.93×	10.27×
laion-clip	OOD	999,448	469,915	4.29×	5.96×	8.59×	11.83×
llama	OOD	255,921	61,275	19.64×	19.34×	19.07×	19.00×
yandex	OOD	1,000,000	62,452	1.18×	1.20×	1.65×	1.85×
yi	OOD	186,843	27,020	2.71×	3.67×	3.28×	3.22×
agnews-mxbai	ID	153,876	15,641	1.39×	1.49×	1.73×	1.86×
arxiv-nomic	ID	268,929	20,107	1.25×	1.42×	1.44×	1.72×
ccnews-nomic	ID	99,066	12,699	1.06×	1.23×	1.34×	1.69×
celeba-resnet	ID	40,320	8,425	1.16×	1.22×	1.29×	1.48×
codesearchnet-jina	ID	274,813	40,473	0.98×	1.26×	1.66×	1.66×
glove	ID	238,503	553,377	1.99×	2.42×	2.55×	2.62×
gooaq-distilroberta	ID	295,005	22,110	1.01×	1.29×	1.36×	1.74×
imagenet-clip	ID	256,233	10,001	1.34×	1.37×	1.63×	1.57×
inaturalist-resnet	ID	99,800	12,138	1.52×	1.57×	1.68×	2.18×
landmark-dino	ID	152,151	14,838	1.28×	1.31×	1.39×	1.62×
landmark-nomic	ID	152,151	7,858	1.44×	1.54×	1.60×	1.54×
simplewiki-openai	ID	52,074	9,699	1.07×	1.35×	1.42×	1.33×
yahoo-minilm	ID	135,461	27,726	1.48×	1.61×	2.08×	1.92×
median				1.37×	1.45×	1.66×	1.80×

C HOW FAR FROM OPTIMAL THE GREEDY IS

The index the greedy of Algorithm 1 builds for the covering problem (7) is also a solution of the budgeted problem (6) at the budget it spends. This appendix measures, on the indices the experiments actually report, how much coverage it gives up against the best possible placement of the same budget.

Instances. We take one index per dataset: the query-aware index of Section 4 at recall@$100 = 0.90$, that is, the configuration of the query-aware method with the lowest latency among those

reaching that recall on the test queries. On out-of-distribution datasets the query-aware method always clusters the learning queries, so only those configurations compete there. Each index fixes a probe count ℓ, a training-recall target α and a clustering (Table 6), and its instance is the problem the greedy solved to build it: the m learning queries with their exact $K = 100$ neighbors, their probe sets $R_\ell(q)$ on that clustering, the demand pairs $\mathcal{D}$ of equation 5 that the primary lists miss, the eligible copies $\mathcal{C}$ that cover them, and the cost $\hat{\pi}_j$ of a copy into list j, the fraction of learning queries that probe j.

The greedy's solution. The greedy adds copies in the order of equation 8, and the index keeps the shortest prefix S_s whose training recall equation 10 reaches α. That prefix spends $B = \hat{\pi}(S_s)$ and covers $f(S_s)$ demand pairs; we report its coverage $f(S_s)/|\mathcal{D}|$, the fraction of the demand pairs it covers.

The LP. Problem (6) at budget B is an integer program with a variable x_c per eligible copy, whether it is bought, and a variable y_{qp} per demand pair, whether it is covered. Its linear relaxation is

$$\max_{x,y} \frac{1}{|\mathcal{D}|} \sum_{(q,p)\in\mathcal{D}} y_{qp} \quad \text{s.t.} \quad y_{qp} \le \sum_{c \text{ covers } (q,p)} x_c, \qquad \sum_{c=(p,j)\in\mathcal{C}} \hat{\pi}_j \, x_c \le B, \qquad x, y \in [0,1]. \tag{15}$$

With x and y restricted to $\{0,1\}$ this is exactly equation 6; relaxing them can only raise the optimum, and the greedy's prefix is a feasible integral point, so

$$\frac{f(S_s)}{|\mathcal{D}|} \;\le\; \mathrm{OPT}_{\mathrm{integral}} \;\le\; \mathrm{OPT}_{\mathrm{LP}},$$

and the ratio greedy/LP is a lower bound on the fraction of the best achievable coverage that the greedy reaches.

The solver. The instances hold up to 161 million demand pairs. We solve equation 15 with HiGHS 1.12 (Huangfu & Hall, 2018), an exact simplex and interior-point solver, called through SciPy 1.18. One HiGHS solve of the whole program takes 17 minutes on the celeba-resnet index at recall 0.95, whose instance has 1.5 million pairs, so for the larger ones we use the program's structure instead. A demand pair concerns one vector, and so do all the copies that cover it, which means only the budget constraint links different vectors. For a price $\lambda \ge 0$ on the budget, the program with that constraint moved into the objective splits into one small LP per vector; we solve every one exactly with HiGHS's dual simplex, in parallel, and search λ by bisection on whether the solution spends more than B. Neither reported bound is taken on trust from the solver. For every λ, weak duality with the row duals u of the per-vector LPs gives the upper bound

$$\mathrm{OPT}_{\mathrm{LP}} \;\le\; \frac{1}{|\mathcal{D}|}\Big(\lambda B + \sum_{(q,p)\in\mathcal{D}} \max(0, 1-u_{qp}) + \sum_{c=(p,j)\in\mathcal{C}} \max\big(0, \textstyle\sum_{(q,p)\,:\,c \text{ covers } (q,p)} u_{qp} - \lambda\hat{\pi}_j\big)\Big),$$

which holds for any $u \ge 0$; and mixing the solutions at the two ends of the final bracket so that they spend exactly B gives a feasible point, whose objective we evaluate directly on the instance. We stop when the two bounds agree to a relative 10^{-9}, which they do on every dataset, and report the upper one. As a check, that single HiGHS solve of the whole program (interior point with crossover, status optimal) returns an optimum inside the two bounds the decomposition gives on the same instance. Each instance is rebuilt for this purpose on the machine of Section 4 and checked against the index measured there.

Probe count 1. There each demand pair has a single covering copy, so equation 15 is a fractional knapsack, and taking copies in decreasing gain per unit cost, which is equation 8, solves it exactly at the budget the prefix spent. The ratio is 1, to within 10^{-9}, on the 4 datasets whose index is solved at $\ell = 1$.

Results. Table 6 gives, per dataset, the greedy's coverage, the LP optimum and their ratio. Over the 24 datasets the ratio has a median of 0.995 and a minimum of 0.9875, on glove. It falls as ℓ grows: the median is 0.995 over the 13 indices solved at $\ell = 2$ and 0.989 over the 7 solved at $\ell = 4$.

Table 6: The greedy against the exact LP optimum of equation 15, for the query-aware index at recall@100 $= 0.90$ on each dataset, at the greedy's own budget. ℓ, α and the clustering ("centroids from") are that index's; on out-of-distribution (OOD) datasets it is always the learning queries. "Demand pairs" is $|\mathcal{D}|$; the greedy and LP columns are fractions of it, the LP column being the verified upper bound, so the last column is a lower bound on the greedy's approximation ratio. Every rebuilt index matches the measured one to the entry.

dataset	queries	ℓ	α	centroids from	demand pairs (M)	greedy coverage	LP optimum	greedy / LP
coco-nomic	OOD	1	0.99	queries	11.36	0.9900	0.9900	1.0000
cqadupstack-lemur	OOD	4	0.85	queries	120.40	0.8341	0.8427	0.9898
cqadupstack-muvera	OOD	4	0.9	queries	132.70	0.8996	0.9076	0.9912
hotpotqa-harrier	OOD	4	0.99	queries	4.88	0.9802	0.9898	0.9903
imagenet-align	OOD	1	0.9	queries	13.83	0.7717	0.7717	1.0000
laion-clip	OOD	2	0.9	queries	92.24	0.8916	0.8945	0.9968
llama	OOD	2	0.95	queries	25.59	0.9500	0.9551	0.9947
yandex	OOD	2	0.9	queries	63.87	0.8434	0.8475	0.9951
yi	OOD	1	0.95	queries	18.66	0.9499	0.9499	1.0000
agnews-mxbai	ID	2	0.9	base set	5.12	0.6993	0.7039	0.9935
arxiv-nomic	ID	2	0.9	base set	9.91	0.7286	0.7325	0.9947
ccnews-nomic	ID	2	0.9	base set	3.55	0.7208	0.7252	0.9939
celeba-resnet	ID	2	0.9	base set	2.17	0.8141	0.8180	0.9953
codesearchnet-jina	ID	4	0.9	base set	7.99	0.6559	0.6632	0.9890
dpr-jina	ID	2	0.9	base set	161.21	0.7398	0.7434	0.9952
glove	ID	4	0.95	base set	12.10	0.9015	0.9128	0.9875
gooaq-distilroberta	ID	2	0.9	base set	10.07	0.7072	0.7116	0.9939
imagenet-clip	ID	2	0.85	base set	6.55	0.4129	0.4144	0.9963
inaturalist-resnet	ID	2	0.85	base set	3.89	0.6149	0.6178	0.9953
landmark-dino	ID	2	0.9	base set	6.39	0.7619	0.7652	0.9958
landmark-nomic	ID	1	0.9	base set	6.85	0.7779	0.7779	1.0000
msmarco-qwen	ID	4	0.9	base set	43.17	0.5904	0.5973	0.9885
simplewiki-openai	ID	2	0.95	base set	1.58	0.8354	0.8430	0.9909
yahoo-minilm	ID	4	0.9	base set	3.57	0.6204	0.6272	0.9891

D Comparison with graph and quantization indices

Section 4 compares placements within one IVF engine. This appendix compares against indices that are not IVF at all, on the same 24 datasets, with eight arms: query-aware IVF and plain IVF as before, HNSW (Malkov & Yashunin, 2020), DiskANN (Subramanya et al., 2019), RoarGraph (Chen et al., 2024), ScaNN (Guo et al., 2020), SOAR (Sun et al., 2023), and query-aware IVF combined with RaBitQ quantization, described below. Within a dataset all eight share one base set, one test-query set and one exact ground truth, and the two IVF arms additionally share their clustering. RoarGraph and query-aware IVF are the two arms that read a query log, and they are given the same learning split, so that comparison holds the side information fixed and varies only what is done with it.

Implementations and parameters. HNSW is FAISS's `IndexHNSWFlat` with $M = 32$ and efConstruction $= 200$; DiskANN is the reference C++ implementation (version 0.7.0) with $R = 64$, $L_{\text{build}} = 100$ and $\alpha = 1.2$; RoarGraph is the authors' implementation with $M_{sq} = 100$, $M_{pjbp} = 35$ and $L_{pjpq} = 500$; ScaNN is version 1.4.2 with $\sqrt{n}$ leaves, asymmetric hashing at two dimensions per block, anisotropic quantization threshold 0.2, and reordering over $\{200, 1000, 5000, 20000\}$ candidates. SOAR is the same ScaNN configuration with spilling switched on ($\lambda = 1.5$ and an over-retrieve factor of 2, the values ScaNN uses when SOAR is enabled), which also stores every point in a second leaf chosen by an orthogonality-amplified residual criterion. ScaNN supports SOAR only under the dot product, so we run it natively on the four inner-product datasets, on the 19 unit-norm Euclidean ones as the equivalent dot product, and on agnews-mxbai, the one that is not unit-norm, through the exact reduction $x \mapsto (x, -\|x\|^2/2)$, $q \mapsto (q, 1)$. On the unit-norm datasets the ScaNN arm therefore scores by squared Euclidean distance and SOAR by the dot product, and ScaNN's dot-product scoring is the more accurate of the two there: on coco-nomic, with otherwise identical settings, it recalls 0.79 against 0.17 at five leaves. Part of SOAR's lead over ScaNN on those datasets is therefore the metric, not the spilling. Parameters are the common settings of each implementation and are not tuned per dataset. Each arm's search knob — efSearch, L, L_{pq}, leaves to search — is swept upward until the arm reaches recall 0.95 or the knob demonstrably stops working, so the maximum reached varies by dataset: efSearch runs to 204,800 and DiskANN's L to 400,000 on the two datasets discussed below, where ordinary values are in the hundreds.

Query-aware IVF with RaBitQ. Query-aware IVF changes only where each vector is stored, so it composes with a compressed representation of the stored vectors. RaBitQ (Gao & Long, 2024) quantizes the residual of a vector from its list's centroid: after a random rotation it keeps one bit per dimension, or B bits in its extension (Gao et al., 2025), together with a few scalar corrections, from which it estimates a query's distance or inner product with a provable error bound. We use FAISS's `IndexIVFRaBitQ` with $B \in \{1, 4\}$ and 4-bit query codes. For each query-aware configuration of Section 4 we read the placement out of the stored query-aware index and re-encode every stored copy as a RaBitQ code against its list's centroid, so a replica costs a code (104 bytes at $d = 768$ and $B = 1$, against 3,072 for a float copy) rather than a vector; one float copy of the base set is kept for re-ranking. A search probes ℓ lists, ranks the codes it scans by the estimated distance, counting each point once, and re-ranks the best $k_f \cdot 100$ exactly, sweeping $k_f \in \{1, 2, \ldots, 32\}$ (to 256 where needed) together with ℓ. Of the 48 (configuration, B) cells per dataset, the 12 that read the fewest bytes per query at recall 0.95 in the recall sweep are timed.

Timing. All arms of a dataset are timed back-to-back in one process, and datasets are timed one at a time. This matters: timing eight datasets concurrently on the same machine inflates plain IVF by $4.08\times$ while leaving query-aware IVF, HNSW and ScaNN unmoved, because the IVF family streams contiguous lists and is memory-bandwidth-bound while the graph arms chase pointers. Parallel measurement would therefore inflate exactly the baseline this paper argues against. SOAR and query-aware IVF with RaBitQ were timed later with the same protocol, on cores of the same machine that time the same configurations within 4% of the core used for the other arms, with no build running; a measurement that overlapped another measurement's index load was repeated. For query-aware IVF with RaBitQ, a configuration was not timed when another timed one reaches at least its recall while reading at most half its bytes (on the 19 datasets first timed without this rule, no configuration it skips lies on the envelope).

Results. Figure 2 gives the recall–QPS envelope of every arm on every dataset and Table 7 the medians at recall 0.9. SOAR, HNSW, ScaNN and RoarGraph are usually faster, at a median $0.29\times$, $0.35\times$, $0.40\times$ and $0.72\times$ the latency of query-aware IVF, while DiskANN is slower, at a median $1.35\times$, with query-aware IVF faster on 14 of the 22 datasets where both reach the target. SOAR is the cheapest arm on 15 of the 24 datasets and a median $1.38\times$ faster than ScaNN; on the four inner-product datasets, where both score by the dot product, it is faster on three. Query-aware IVF is the cheapest arm on 2 of the 24 datasets, yi and llama, where HNSW and DiskANN never reach recall 0.9: on yi, HNSW returns recall 0.298 from the same 30,793 candidates at efSearch of 51,200, 102,400 and 204,800, three doublings that buy nothing, and DiskANN freezes at 0.696 for L up to 400,000.

Two further observations qualify the table. First, the gap depends on the metric: counted in candidates the graph arms look three to six times cheaper, but in wall-clock time the gap narrows to at most three, and DiskANN falls behind, because a scanned candidate in a contiguous IVF list is much cheaper than a pointer chase. A comparison of distance computations alone would overstate the graph indices. Second, the IVF-family result of Section 4 is unchanged in this setting, and it is what this paper claims: query-aware IVF is a median $2.00\times$ faster than plain IVF, winning on all 24 datasets.

Query-aware IVF with RaBitQ. Storing RaBitQ codes instead of the vectors makes query-aware IVF a median $2.7\times$ faster at recall 0.9 ($0.37\times$ the latency of the uncompressed index, Table 7), and faster on 16 of the 24 datasets. The gain is concentrated on the in-distribution datasets, a median $0.27\times$ the latency against $1.01\times$ on the out-of-distribution ones. Of the eight datasets where it is slower, the extremes are llama and yi ($d = 128$), $7.4\times$ and $5.6\times$ slower, where re-ranking the shortlist alone costs more than scanning the full vectors did. Against the other arms, the combination is a median $1.14\times$ faster than ScaNN, $2.49\times$ faster than DiskANN and $6.08\times$ faster than plain IVF, while HNSW and RoarGraph are a median 9–10% faster than it and SOAR $1.41\times$ faster. It is the fastest of the eight arms on 3 of the 24 datasets (simplewiki-openai, cqadupstack-muvera and agnews-mxbai), and query-aware IVF with or without RaBitQ is the fastest on 5, against 2 without it. The combination is deliberately simple—an existing quantizer applied to the query-aware placement, neither tuned to the other—so the placement gain carries over to a compressed index.

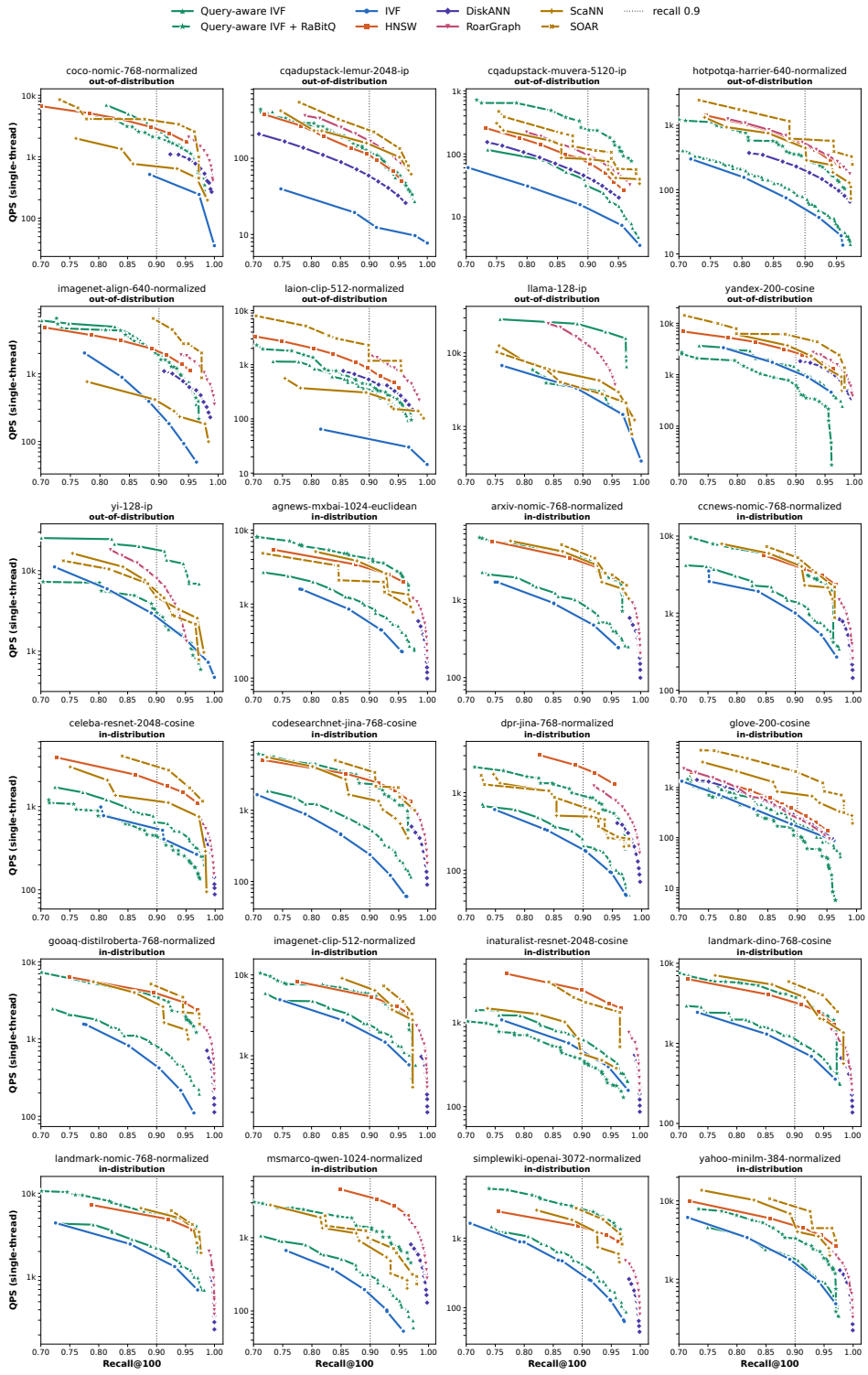


Figure 2: Single-thread QPS (log) against recall@100 for all eight arms, one panel per dataset, out-of-distribution datasets first; each arm is its own envelope, query-aware IVF with RaBitQ and SOAR are dashed in the colors of query-aware IVF and ScaNN, and the dotted line marks recall 0.9. HNSW and DiskANN stop short of the target on yi and llama, the two datasets where query-aware IVF is the cheapest arm.

Table 7: Eight arms at recall@100 = 0.9 on the 24 VIBE datasets, each read through its lower envelope. "Reaches" counts the datasets where the arm attains the target; "query-aware wins" counts the datasets, among those where both arms reach it, on which query-aware IVF is faster. Ratios above 1 favor query-aware IVF. Build times of vanilla and query-aware IVF are those of Table 2, one index each; the other arms' come from this comparison's run on a shared machine. No build includes the exact neighbors of the learning queries, which query-aware IVF and RoarGraph both consume. A dagger marks arms whose build time comes in part from the builder's own log (see the text). ScaNN, SOAR and query-aware IVF with RaBitQ score most points against quantized codes and compute exact distances only for a short re-ranking list, so they have no candidate count comparable to the other arms. For query-aware IVF with RaBitQ, "query-aware wins" counts the datasets on which the uncompressed query-aware IVF is faster, and its build time is one query-aware index as above plus the RaBitQ encoding of its fastest cell that reaches the target.

	reaches	query-aware	median ratio to query-aware		median
arm	0.9	wins	latency	candidates	build (s)
IVF	24/24	24/24	2.00×	2.25×	5
HNSW	22/24	2/22	0.35×	0.16×	200
DiskANN	22/24	14/22	1.35×	0.31×	191†
RoarGraph	24/24	3/24	0.72×	0.26×	455†
ScaNN	24/24	7/24	0.40×	—	38
SOAR	24/24	2/24	0.29×	—	35
query-aware IVF	24/24	—	1.00×	1.00×	10
query-aware IVF + RaBitQ	24/24	8/24	0.37×	—	22

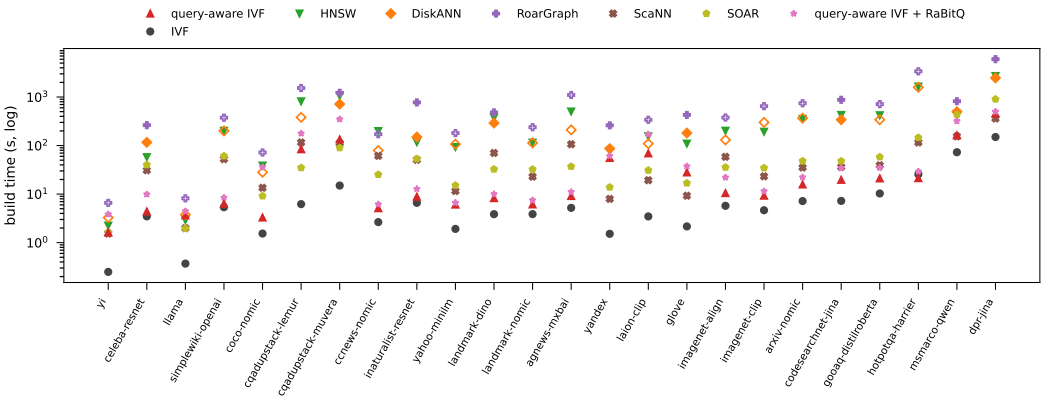


Figure 3: Build time of every arm on every dataset, log scale, datasets ordered by n. One index per arm, without the learning queries' exact neighbors, as in Table 7; vanilla and query-aware IVF are timed as in Table 2. Open markers are the 15 DiskANN and RoarGraph builds whose time comes from the builder's own log rather than from our harness, which skips a build when the index is already on disk; where both exist they agree within 1–4%.

Build time. Figure 3 and Table 7 give build times. As in Section 4, no build includes computing the exact neighbors of the learning queries, which query-aware IVF and RoarGraph both consume. One query-aware index then takes a median 10 s, against 5 s for plain IVF, 35 s for SOAR, 38 s for ScaNN, 191 s for DiskANN, 200 s for HNSW and 455 s for RoarGraph; one query-aware index plus the RaBitQ encoding of one cell takes a median 22 s. For DiskANN and RoarGraph, 15 of the 24 times come from the builder's own log, which agrees with our timer to within 4% where both exist.